|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Computer Science and Engineering**  **Department of Computer Science and Artificial Intelligence** | Logo, company name  Description automatically generated | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم وهندسة الحاسب**  **قسم علوم الحاسب والذكاء الاصطناعي** | |  |  |

|  |
| --- |
| **Lab 3** |
| **CCAI 312 Pattern Recognition** |
| **Third Trimester 2023**   |  |  | | --- | --- | | **Lab Date/Time:**  **Lab assignment submission Date/Time:** |  | | **Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | |

|  |  |
| --- | --- |
| **Instructor Name** | **Section** |
|  |  |

**Instructions**:

The lab assignments must be submitted before the allocated Date/Time.

The lab assignments must by uploaded on LMS / sent by email to teacher@uj.edu.sa.

Plagiarism will be punished according to university rules.

|  |  |
| --- | --- |
| **PLO/CLO** | **SO** |
| **PLO S2 (CLO 2):** **Implement** a suitable pattern recognition technique for a given problem using Python | **SO 2:** Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program’s discipline |
| **PLO C4 (CLO 3):** Discover and apply new knowledge as needed, using appropriate learning strategies. | **SO 7:** An ability to acquire and apply new knowledge as needed, using appropriate learning strategies. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Max Score** | **Student Score** |
| **PLO S2 / CLO 2 / SO 2** | **Task 1** | **2** |  |
| **PLO C4 / CLO 3 / SO 7** | **Task 2** | **2** |  |
| **Total** | |  |  |

# Lab Description

# In this lab you will train and evaluate k-Nearest Neighbours models to perform network intrusion detection and athlete selection prediction.

using two publicly available datasets.

# Objectives

* Have a working knowledge of k-Nearest Neighbours models.
* Utilizing k-Nearest Neighbors to develop and assess a network intrusion detection and athlete selection prediction model.
* Learn how to perform hyper parameter tuning.

# Lab Tool(s)

<https://www.kaggle.com/>

or

<https://colab.research.google.com/>

# Lab Deliverables

Submit A notebook to Blackboard containing your solution to the lab assessment at the end of this document.

# References:

<https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection>

<https://learning.oreilly.com/library/view/supervised-learning-with/9781484261569/>

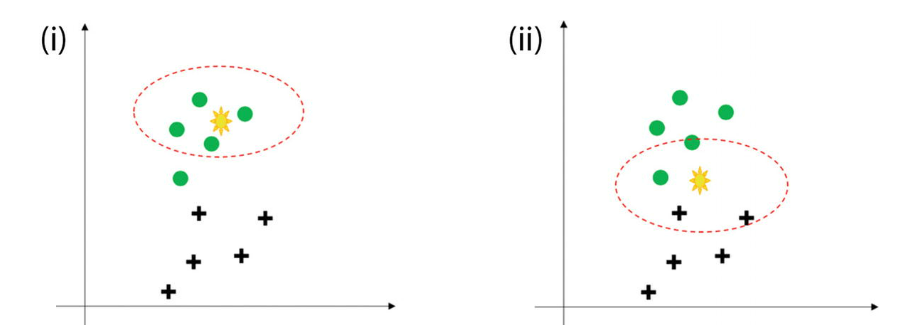
Other Resources:

<https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm>

# k-Nearest Neighbours

# **1-Introducing k-Nearest Neighbours**

Knn is one of the most popular ML techniques where the learning is based on the similarity of data points with each other. knn is a **nonparametric** model; it does not construct a “model” and the classification is based on a simple majority vote from the neighbors. It can be used for classification where the relationship between attributes and target classes is complex and difficult to understand, and yet items in a class tend to be fairly homogenous on the values of attributes. But it might not be the best choice for an unclean dataset or where the target classes are not distinctively clear. If the target classes are not clearly demarcated, then it leads to an obvious confusion while taking the majority vote. knn can also be used for **regression** problems to make a prediction for a continuous variable. In the case of regression, the final output will be the average of the values of the neighbors and that average will be assigned to the target variable.



The steps which are followed in k-nearest neighbour are as follows:

1.We receive the raw and unclassified dataset which has to be worked upon.

2.We choose a distance matrix from Euclidean, Manhattan or Minkowski.

3.Then calculate the distance between the new data points and the known classified training data points.

4.The number of neighbors to be considered is defined by the value of “k”.

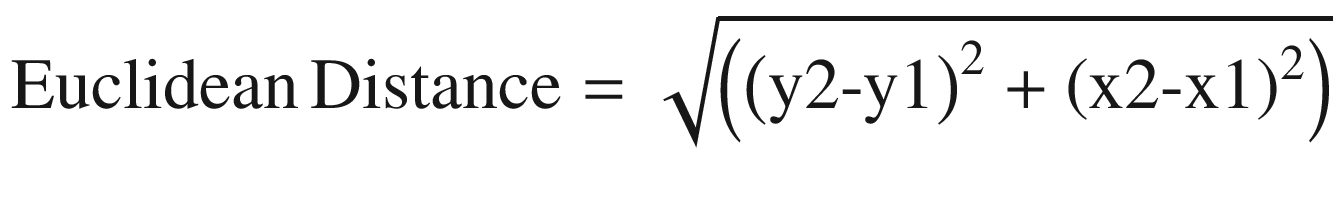
5.It is followed by comparing with the list of classes which have the shortest distance and count the number of times each class appears.

6.The class with the highest votes wins. This means that the class which has the highest frequency and has appeared the greatest number of times is assigned to the unknown data point.

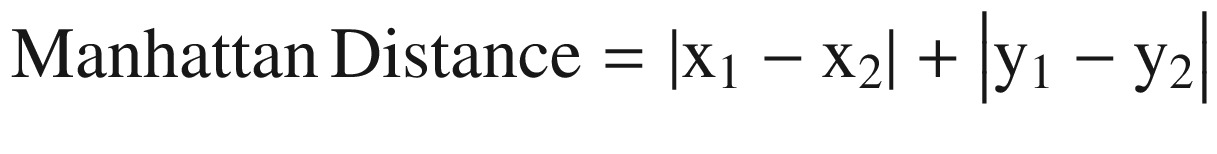
**Tip**  
Parametric models make some assumptions about the input data like having a normal distribution. However, nonparametric methodology believes that data distributions are undefinable by a finite set of parameters and hence do not make any assumptions.

Popular distance matrices used are

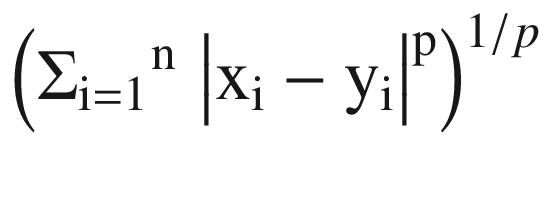
1. **Euclidean Distance** : probably the most common and easiest way to calculate between two points. It is square root of the sum of the squares of distances:



1. **Manhattan Distance** : The distance between two points measured along axes at right angles. Sometimes it is also referred to as city block distance. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is



1. **Minkowski Distance** : This is a metric in a normed vector space. Minkowski distance is used for distance similarity of vectors. Given two or more vectors, find the distance similarity of these vectors. Mainly, the Minkowski distance is applied in ML to find out the distance similarity. It is a generalized distance metric and can be represented by the following formula:



* + 1. where by using different values of p we can get different values of distances. With the value of p = 1 we get Manhattan distance, with p = 2 we get Euclidean distance, and with p= ∞ we get Chebychev distance.

1. **Cosine Similarity**: It is a measure of similarity between two nonzero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any angle in the interval (0,π] radians.

## **1.2 Case Study: Network intrusion detection using k-Nearest Neighbor**

# **Dataset**

# The dataset to be audited was provided which consists of a wide variety of intrusions simulated in a military network environment. It created an environment to acquire raw TCP/IP dump data for a network by simulating a typical US Air Force LAN. The LAN was focused like a real environment and blasted with multiple attacks. A connection is a sequence of TCP packets starting and ending at some time duration between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Also, each connection is labelled as either normal or as an attack with exactly one specific attack type. Each connection record consists of about 100 bytes. For each TCP/IP connection, 41 quantitative and qualitative features are obtained from normal and attack data (3 qualitative and 38 quantitative features) .The class variable has two categories: • Normal • Anomalous

**Step 1**: Import all the required libraries. We are importing

pandas,numpy,matplotlib, seaborn.

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline

**Step 2**: Import the dataset using read.csv method from pandas. Let’s have a look at the top five

network\_data= pd.read\_csv('/content/Network\_Intrusion.csv')  
network\_data.head()

**Step 3**: Now we will do the regular checkup of the data using info() and describe command.

network\_data.info()

network\_data.describe().transpose()

**Step 4**: Now check for the null values. In our dataset there are no null values fortunately.

network\_data.isnull().sum()

**Step 5**: Let’s have a look at the class distribution. And we will visualize it too.

network\_data["class"].value\_counts(normalize=True)

pd.value\_counts(network\_data["class"]).plot(kind="bar")

**Step 6:** There are a few categorical variables in our dataset. We have to convert them to numerical variables using one-hot encoding.

from sklearn.preprocessing import LabelEncoder  
label\_encoder = LabelEncoder()  
network\_data['class'] = label\_encoder.fit\_transform(network\_data['class'])  
network\_data['protocol\_type'] = label\_encoder.fit\_transform(network\_data['protocol\_type'])  
network\_data['service'] = label\_encoder.fit\_transform(network\_data['service'])  
network\_data['flag'] = label\_encoder.fit\_transform(network\_data['flag'])

**Step 7**: One-hot encoding increases the number of variables in the dataset. Let’s see the number of columns in the dataset with added variables:

network\_data.columns

**Step 8**: Next we will standardize our dataset by using StandardScaler in scikit learn.

from sklearn import preprocessing  
from sklearn.preprocessing import StandardScaler  
X\_std = pd.DataFrame(StandardScaler().fit\_transform(network\_data))  
X\_std.columns = network\_data.columns

**Step 9**: Now it is the time to split the data in train and test. We are dividing the data in an 80:20 ratio.

import numpy as np  
from sklearn.model\_selection import train\_test\_split  
  
X = np.array(network\_data.iloc[:, 1:5]) #Transform data into features  
y = np.array(network\_data['class']) #Transform data into targets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)

**Step 10**: The sklearn.cross\_validation has deprecated and hence we received this error. Again, try to split in train and test using sklearn.model\_selection.

from sklearn.model\_selection import train\_test\_split  
# Transform data into features and target  
X = np.array(network\_data.iloc[:, 1:5])  
y = np.array(network\_data['class'])  
# split into train and test  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)

**Step 11**: Print the shape of the data by print(X\_train.shape).

print(y\_train.shape)

**Step 12**: Print the shape of the test data by print(X\_test.shape).

print(y\_test.shape)

**Step 13**: We will now train the model using training data and iterate with different values of k=3,5,9.

# With k = 3

from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import recall\_score  
# instantiate learning model (k = 3)  
knn\_model = KNeighborsClassifier(n\_neighbors = 3)  
#Fitting the model  
knn\_model.fit(X\_train, y\_train)  
y\_pred = knn\_model.predict(X\_test) # predict the response  
print(accuracy\_score(y\_test, y\_pred)) # Evaluate accuracy

The answer is (…………………………………………)

# With k = 5

knn\_model = KNeighborsClassifier(n\_neighbors=5)   
knn\_model.fit(X\_train, y\_train) # Fitting the model  
y\_pred = knn\_model.predict(X\_test) # Predict the response  
print(accuracy\_score(y\_test, y\_pred)) # Evaluate accuracy

The answer is (…………………………………………)

With k = 9

knn\_model = KNeighborsClassifier(n\_neighbors=9)  
knn\_model.fit(X\_train, y\_train) # Fitting the model  
y\_pred = knn\_model.predict(X\_test) # Predict the response  
print(accuracy\_score(y\_test, y\_pred)) # Evaluate accuracy  
The answer is 0.9867037110537805

The answer is (…………………………………………)

**What is the best k value?** (…………………………………………)

**Step 14**: We have tested with three values of k. We will now iterate on multiple values of k. We will run the knn with the no. of neighbors to be 1,3,5…19 and then find the optimal number of neighbors based on the lowest misclassification error.

k\_list = list(range(1,20)) # creating odd list of K for KNN  
k\_neighbors = list(filter(lambda x: x % 2 != 0, k\_list)) # subsetting just the odd ones  
ac\_scores = [] # empty list that will hold accuracy scores  
# perform accuracy metrics for values from 1,3,5....19  
for k in k\_neighbors:  
    knn\_model = KNeighborsClassifier(n\_neighbors=k)  
    knn\_model.fit(X\_train, y\_train)  
    y\_pred = knn\_model.predict(X\_test)   # predict the response  
    scores = accuracy\_score(y\_test, y\_pred)  # evaluate accuracy  
    ac\_scores.append(scores)  
# changing to misclassification error  
MSE = [1 - x for x in ac\_scores]  
# determining best k  
optimal\_k = k\_neighbors[MSE.index(min(MSE))]  
print("The optimal number of neighbors is %d" % optimal\_k)

**Step 15**: Let’s print the impact of different values of k on the misclassification error.

import matplotlib.pyplot as plt  
# plot misclassification error vs k  
plt.plot(k\_neighbors, MSE)  
plt.xlabel('Number of Neighbors K')  
plt.ylabel('Misclassification Error')  
plt.show()

**Step 16**: It turns out that k=3 gives us the best result. Let’s implement it:

#Use k=3 as the final model for prediction  
knn = KNeighborsClassifier(n\_neighbors = 3)  
# fitting the model  
knn.fit(X\_train, y\_train)  
# predict the response  
y\_pred = knn.predict(X\_test)  
# evaluate accuracy  
print(accuracy\_score(y\_test, y\_pred))  
print(recall\_score(y\_test, y\_pred))

Task **Task 1: [PLO S2 / CLO 2 / SO 2] [2 marks]**

1. First import required libraries, load “**Athlete Selection**” dataset into a data frame and print the first 5 rows.
2. Create a new variable “names” that contain the dataframe indexes and print it.
3. Store features and labels in numpy arrays X and y and print the first feature of the first example. (Hint: Use **pop** method)
4. Fit NearestNeighbors sickit\_learn model to the data with **K=2** and **radius=0.4**.

NearestNeighbors is Unsupervised learner for implementing neighbor searches

1. Get parameters of this model (also called estimator)
2. Find k nearest neighbors of a point by returning the distances (array) and the indices(array) of the nearest k points

q1 = [3.25,8.25] # query point, equivalent to x3

q2 = [0.2, 3.3]

1. What does the following code do?

q = [5.0,7.5]  
q3n = athlete*\_neigh.kneighbors([q], n\_*neighbors = 3)[1][0]  
for n in q3n:  
  print(names[n])

1. Fit KNeighborsClassifier sickit\_learn model to the data with K=3.

KNeighborsClassifier is classifier implementing the k-nearest neighbors vote.

1. Evaluate the model Using training data as test set (Hint: Use model predict method)
2. How accurate is the model? Is the accuracy good or bad? Why do you believe the model was so accurate?

**Task? 2: [PLO C4 / CLO 3 / SO 7] [2 mark]**

1. Read athlete\_test file and store features and labels in numpy arrays X\_test and y\_test (Hint: Use pop method)
2. Fit KNeighborsClassifier sickit\_learn model to the data with K=3.
3. Evaluate the model Using X\_test and y\_test data as test set (Hint: Use model predict method)
4. Use StandardScaler from sklearn to map features values to unit variance and fit and evaluate a new KNeighborsClassifier model
5. Use MinMaxScaler from sklearn to map features values to unit variance and fit and evaluate a new KNeighborsClassifier model
6. Evaluate previous three models with different values of K
7. Which of the models is most accurate?